Web 2.0-Based Social Informatics Data Grid

Wenjun Wu
University of Chicago & Argonne National Laboratory,
9700 S Cass Ave, Argonne, IL 60439
wwj@ci.uchicago.edu

Thomas Uram
University of Chicago & Argonne National Laboratory,
9700 S Cass Ave, Argonne, IL 60439
turam@ ci.uchicago.edu

Michael E. Papka
University of Chicago & Argonne National Laboratory,
9700 S Cass Ave, Argonne, IL 60439
papka@ ci.uchicago.edu

ABSTRACT

The Social Informatics Data Grid (SIDGrid) is a new cyberinfrastructure designed to transform how social and behavioral scientists collect and annotate data, collaborate and share data, and analyze and mine large data repositories. The major design goals for the SIDGrid are to integrate those commonly used social and behavior science tools and provide researchers an easy-to-use web interface to run these data intensive applications efficiently on TeraGrid resources. SIDGrid is also a collaborative environment where scientists can share experimental data and analysis results with their team members.

OpenSocial, a social networking framework initiated by Google, provides a Web 2.0 approach to integration of web applications and building collaborative cyber environments. Using OpenSocial, we present a new application framework for SIDGrid that enables scientists to define their analysis tools in XML and generate application gadgets as a Web 2.0 interface for running analytical workflows on TeraGrid. Based on this framework, we have developed a new SIDGrid science gateway to improve the user's experience and simplify SIDGrid application management and development of collaborative web applications.

Categories and Subject Descriptors

H.3.5 [**Online Information System**]: Web-based Services

General Terms

Design

Keywords

OpenSocial, Gadget, Web 2.0, TeraGrid, Social and Behavioral Science.

1. INTRODUCTION

The Social Informatics Data Grid (SIDGrid) [1] is a new cyberinfrastructure designed to transform how social and behavioral scientists collect and annotate data, collaborate and share data, and analyze and mine large data repositories. The grand challenge in social and behavioral science is how to model human behavior as a dynamic, multicausal system that occurs over multiple time scales. To meet this challenge, behavioral scientists must store multiple measures of neural, cognitive, and social behaviors of humans into a common database so that they can access and analyze these measures at multiple levels simultaneously in a collaborative way.

Because of the diversity of the datasets in the social and behavioral sciences, researchers need to utilize numerous analysis tools to retrieve the features from text, images, audio, video, and sensor data. Moreover, analysis tasks for behavioral science research usually require extensive computational resources provided by Grid-enabled problem solving environments such as TeraGrid. The major design goals for the SIDGrid are to integrate those commonly used social and behavioral science tools and provide researchers an easy-to-use web interface to run these application tools over their massive datasets efficiently on TeraGrid resources. SIDGrid also seeks to provide a collaborative environment where scientists can share their experimental data and analysis results with their team members.

To achieve these goals, we need an application framework for users to easily integrate new analysis tools into the SIDGrid and expose these tools as web services. Such a framework should allow users to describe their scientific applications and generate web interfaces for these applications automatically. The framework should also efficiently manage and execute computational workflows for the applications on the TeraGrid resources. Although considerable research [2][3] has explored ways to generate application-specific user interfaces, wrap scientific applications as web services, and run these services on Grid environments, we believe that new exploration should be conducted to find out how Web 2.0 technologies can be applied to the same problems with innovative solutions.

OpenSocial [4], a social networking framework initiated by Google, presents a Web 2.0 approach to the integration of web applications and the construction of collaborative cyber environments. In OpenSocial, every web application is regarded as a gadget, which can define its HTML content and control logic in client-side JavaScript. OpenSocial also provides a social data API to access information about people, their friends, and their data, within the context of an OpenSocial container.

OpenSocial gadgets as client-side web applications are ideal candidates for the rich web interface of the SIDGrid applications. Using OpenSocial, therefore, we have developed a new application framework for SIDGrid and have used the framework to construct a SIDGrid science gateway that can improve user experience, simplify the SIDGrid application management, and promote scientific collaborations among SIDGrid users.

This paper is organized as follows. Section 2 gives an overview of the SIDGrid framework. Section 3 presents the details about the application and workflow management of the framework. In Section 4, we discuss how to use the OpenSocial framework to manage SIDGrid applications gadgets and build collaborative gadgets based on the SIDGrid data model. Section 5 summarizes our work and our conclusions.

2. SIDGrid Web 2.0 Framework Overview

We selected OpenSocial as the basis for the SIDGrid Web 2.0 framework. It standardizes the practices of both gadget programming and online social networking, enabling web developers to write social gadgets that can run in any OpenSocial-compliant container. To have the SIDGrid server become a host environment for OpenSocial gadgets, we developed an application framework that implements all the necessary services, including gadget rendering and access to social data .

Specifically, we reused an application description tool named Mobyle [5] from the bioinformatics community for SIDGrid users to describe the command-line syntax for their applications. From the Mobyle XML description, the SIDGrid framework can generate application gadgets that can be flexibly integrated into OpenSocial-compliant web sites such as iGoogle and MySpace. For the workflow management, the Swift [6] workflow engine is used to run SIDGrid application workflows on TeraGrid; we chose Swift because it is designed for loosely coupled, data-intensive computing on Grid environments.

Figure 1 shows the basic structure of the SIDGrid science gateway. The major components in the SIDGrid application framework consist of five functional subsystems: the application management module, the gadget hosting module, the SIDGrid web services module, the workflow execution module, and the SIDGrid data management module.

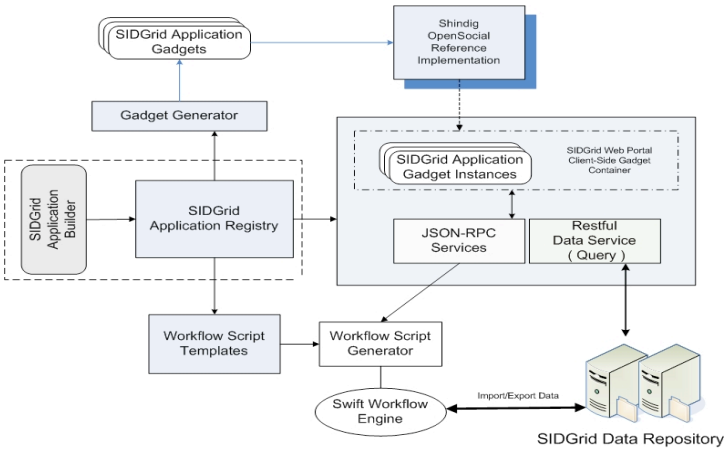

Figure 1. SIDGrid Web2.0 framework.

In the application management module, the application registry keeps the metainformation of SIDGrid application tools, which lists the command-line syntax, the installation locations, and the run-time requirements of all the deployed SIDGrid applications. We customized Mobyle, which is an XML description schema designed to specify the command-line syntax of bioinformatics tools, for the description of the SIDGrid applications. Through the SIDGrid application builder page, a user can define the command-line format for his program in the Mobyle XML file, preview the web interface created by the gadget generator from the XML file, and import the interface to the SIDGrid.

The gadget hosting module renders the generated application gadgets and provides the gadget instances proxy services to fetch web contents from the SIDGrid or other external data sources. Shindig [7], an OpenSocial reference implementation, becomes the foundation for the development of this module. A more detailed description of Shindig and the OpenSocial software stack can be found in Section 4.

Clearly, SIDGrid gadgets need more SIDGrid-specific services besides the standard gadget I/O services from Shindig.

GCE'09, Nov 20, 2009, Portland, Oregon, USA.

All the SIDGrid-specific communication services are available through the SIDGrid web services module. They offer RESTful web services and the JSON-RPC service to those generated application gadgets to create workflow instances, search for the relationship between SIDGrid users, and query the SIDGrid experiment data.

The workflow execution module is built on top of the Swift package, which is a workflow system for large-scale, loosely coupled parallel computation. It enables behavior science analysis applications to process large datasets in a parallel way. Each user's request for SIDGrid to run an application on his data files is translated into a Swift script. The flexible file mapping mechanism in Swift enables the user to run the analysis on either a single data file or multiple data files.

A workflow request is first recorded in a persistent database that is polled by the Swift engine daemon. This daemon executes the workflows by running a shared Swift engine instance for all of them. Because the Swift engine has a weighted scheduling mechanism for the job execution, the daemon can dynamically decide the best TeraGrid resources that are available to run the SIDGrid workflows and can resubmit any failed jobs. Based on the log files produced by the Swift engine during the execution of the workflows, the Swift daemon monitors their progress and updates their status in the workflow database, which can be visualized on the SIDGrid workflow monitor gadget.

The SIDGrid data model is designed for behavior and social science research. The basic management unit is a SIDGrid experiment, which usually consists of multiple video and audio files, annotation data, timer-series files, and linked files. On the basis of this data model, the SIDGrid data management module allows users to create their SIDGrid experiments, to import and export research data files, to add semantic tags for the data, and to query the data files.

3. SIDGrid Scientific Application and Workflow Management

In this section we present an example of using the SIDGrid Web 2.0 framework to define applications, create application gadgets, and run workflows. The science application in this example is Praat [8], a popular tool for speech analyses such as spectrum, pitch, and intensity in phonetics. Some SIDGrid users from the domain of linguistics research use Praat to carry out experiments on prosodic analysis.

3.1 Application XML Description

Mobyle defines an XML schema that can be used to describe any command-line program. It allows users to specify the type, name, and format of each argument in a flexible way. Short python or perl code snippets can be inserted into a Mobyle description to express the dependence of the arguments and generate the appropriate argument format. Figure 2 displays a segment of the Mobyle XML description for Praat.

```
 <parameters>
  <parameter ismandatory="1" issimple="1" ismaininput="1">
   <name>inscript</name>
   <prompt lang="en">praat script file</prompt>
   <type>
              <datatype>
              <class>File</class>
              </datatype>
             </type>
             <format>
               <code proglang="python"> ("",str(value))[value is not
None]</code>
    </format>
   <argpos>1</argpos>
  </parameter>
  <parameter ismandatory="1" issimple="1" ismaininput="1">
   <name>infile</name>
   <prompt lang="en">praat data wave file</prompt>
   <type>
              <datatype>
              <class>File</class>
              </datatype>
             </type>
             <format>
  <parameter ismandatory="1" issimple="1" isoutput="1">
   <name>outfile</name>
   <prompt lang="en">praat transformation output file</prompt>
   <type>
              <datatype>
              <class>File</class>
              </datatype>
             </type>
             <format>
               <code proglang="python"> ("",str(value))[value is not
None]</code>
    </format>
   <argpos>1</argpos>
  </parameter>
 </parameters>
</program>
```

Figure 2. Praat XML description.

SIDGrid users who know Mobyle well and have a complex command-line syntax to define can directly write down their XML file and upload to the SIDGrid. In most cases, however, users do not wish to spend time learning the Mobyle and prefer to describe their applications through a simple web interface. We developed such a builder page (see Figure 3), called the SIDGrid application builder, so that users can define the command-line format for their programs in the Mobyle XML, preview the web interface generated by the gadget generator from the XML file, and import the interface to the SIDGrid.

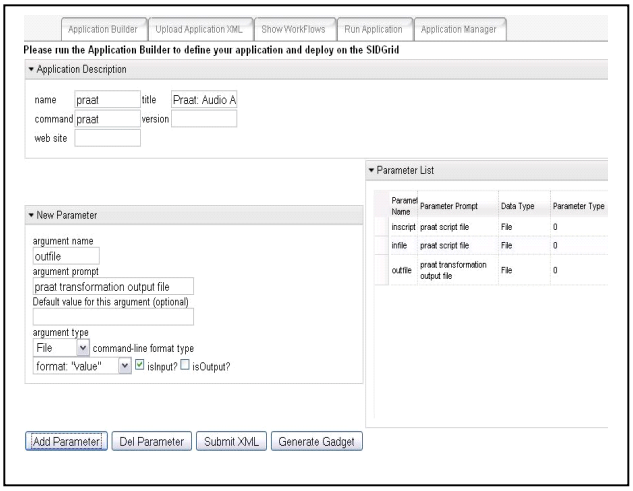

Figure 3. SIDGrid application management page.

3.2 Application Gadget Generator

The gadget generator can create an OpenSocial gadget that consists of the gadget metadata, as well as HTML markups and JavaScript codes, from the Mobyle XML description of each SIDGrid application by using XSLT templates. The XSLT template in the generator transforms the application XML into an HTML snippet with a predefined CSS style sheet. A JavaScript template is used to produce JavaScript code for OAuth [9] security handling, parameter marshalling, and invocation of the generic JSON-RPC service. The JavaScript code segment for the Praat gadget is illustrated in Figure 4.

```
function runPraat( ){
  // create a JSONRpcClient object
  jsonrpc = new JSONRpcClient("/SIDGrid/JSON-RPC");
  var params = new Object();
  // java class hint
  params.javaClass = 'java.util.Hashtable';
  params.map = {};
  params.map[‘inscript] = document.getElementById("inscript
").value;
  params.map['infile’ ] =
document.getElementById("infile").value;
  params.map['outfile'] =
document.getElementById("outfile").value;
  // Hashtable params
  result = jsonrpc.JobService.run("praat", params);
```

Figure 4. JavaScript code in the generated Praat gadget.

3.3 Application Service Deployment

After completing the definition of an application, the user will face another problem: how to deploy the executable application package for this application onto the TeraGrid resources and set up an appropriate transformation registry entry in the Swift engine to invoke the installed package. It is still tricky to achieve automatic deployment of any application under the TeraGrid community account allocation [10] without involving the administrators of science gateways. Therefore, instead of allowing users to install application packages dynamically, we introduce a collaborative mechanism between administrators and users to work together to deploy the application. When a user finishes a new application XML description in the SIDGrid Application Builder, he can request the administrator to deploy this new application on TeraGrid. Following the URL to the application package, the administrator can download the package, install it, and test it on multiple TeraGrid clusters. After the application is successfully deployed, the gadget and RPC service for this application will be activated and made available for users.

3.4 Generic JSON-RPC Service and Workflow Script Generator

A generic JSON-RPC service is designed to handle AJAX requests from SIDGrid gadgets for workflow operations including initiation, status query and result retrieval. Figure 5 shows four major methods of the service.

```
Interface Generic JSONRPC Service
{
String WorkflowID runWorkflow(String application,
Hashmap<String,String> params);
String checkStatus(String workflowId);
// get the rest links to the output files
List<URL> getResults(String workflowId);
String stopWorkflow(String workflowId);
}
```

Figure 5. Generic RPC service interface.

The service unmarshals RPC requests from gadgets and calls both database handlers to initializes the workflow object in the database and a workflow script generator to compose workflow scripts for the Swift engine to execute on the TeraGrid resources. For example, when a user launches the Praat gadget to run a pitch analysis on a audio wave file, the service gets the input parameters, including the input script file path, the input audio file path, and the output file name. Given the input parameters for the run, the script generator parses the Mobyle XML of the application, formats the command for this application, and produces a Swift script for this run (Figure 6). This script consists of four parts: a file type declaration, a Praat transformation procedure that contains the Praat command line, a group of file mappings linking the logical file variables with physical data files, and the final statement that calls the transformation.

```
type File
(File outfile) runpraat(File inscript, File infile){
  praat  @inscript @infile @outfile
}
## file mapper
File inscript <single_file_mapper;file="@exp/script">;
File infile <single_file_mapper;file="@exp/input">;
File outfile <single_file_mapper;file="@exp/output">;
outfile = runpraat(inscript, infile);
```

Figure 6. Generated Swift script for running Praat.

Because social and behavioral science data are collected from human beings, data security is an important concern. To protect

the privacy of studied subjects, researchers usually do not want to make their data available to the public. It is essential, therefore, to set up a private OpenSocial container for rendering the SIDGrid gadgets to alleviate the privacy concern.

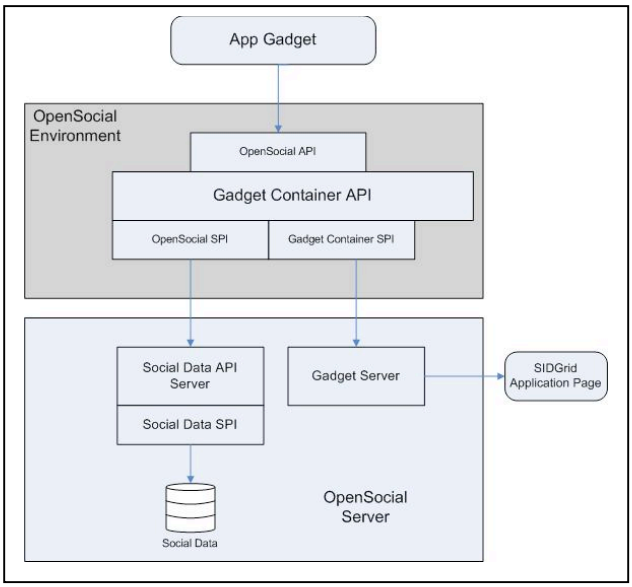

Figure 6. OpenSocial container stack

Figure 6 illustrates the major components in an OpenSocial container. A gadget server can parse a gadget XML file and render this gadget into HTML and JavaScript codes consisting of user codes and the OpenSocial supporting library. The gadget server also provides communication service to gadgets, especially cross-domain proxy to support gadgets.io.makerequest. The Social Data API Server acts as an interface for OpenSocial gadgets to access the social data provided by external data source. Through this interface, a gadget can retrieve a user's profile, find friends for a user, and display activities for users.

We note that Shindig is not a full-fledged OpenSocial container because it still has no services such as gadgets layout, gadgets management, and security. By using the building blocks in the Shindig's package, however, we can easily transform the SIDGrid dynamic web pages into a client-side OpenSocial gadget container. We also developed the server-side code to handle the gadget management and regular web security that are built on top of HTTPS and user-password based authentication.

4.1 Gadget Rendering

Figure 7 shows SIDGrid ExperimentView page that presents users the details of a selected experiment including ownership and data files. On the page, users can pick up files and choose transformations to run on them. In Figure 7, the Praat transformation and the pitch script are selected to run on the wave file elanexample1.wav. The Praat UI is displayed as a Praat gadget loaded within an iframe in the page. This iframe, which isolates the JavasScript code in the gadget from the container page, needs an src attribute set to a URL pointing to the gadget renderer of the server. Here the gadget renderer is the embedded Shindig server in the SIDGrid.

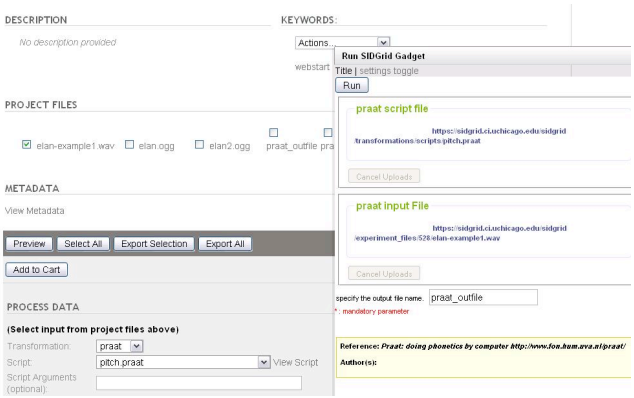

Figure 7. SIDGrid gadget container page.

This URL link has an important parameter called the security token. This is a short-lived token that has encoded in it all the necessary information about the site, gadget, and viewer. Once a SIDGrid gadget is initialized, it parses the security token in the URL and reads the OwnerID field as a user ID for calling SIDGrid JSON-RPC application services.

Using the security token raises a security issue, however. The Shindig package provides no authentication mechanism to prevent malicious users from faking a security token with a user identity of others and accessing the SIDGrid services on the identity. Therefore, we must encrypt the security token and restrict the access endpoints to the Shindig server. The Apache Shindig package supports encrypted security token, which requires a shared key file between the gadget container page and the Shindig server. By using this shared key, the SIDGrid container page creates an encrypted security token for any gadget hosted in the container. Shindig's gadget renderer can decode the security token and then pass the token to the initialized gadget instance. We can also place the access control list for the gadget rendering. Currently the SIDGrid Shindig server accepts the rendering request from only a small number of hosts, including the SIDGrid server.

Another issue is the RPC communication between the application gadget and the container page. SIDGrid application gadgets have to communicate with the gadget container page to retrieve the context information such as current experiment ID and selected data files that this application starts to analyze. As shown in Figure 8, the Praat gadget instance has to know the pitch analysis script and audio wave file from the ExperimentView page. In the SIDGrid, however, the communication between gadgets and container web pages become a cross-domain JavaScript RPC. Being loaded in an iframe of the container page, a gadget has to follow the JavaScript sandboxing and same-domain security policy, thereby prohibiting its JavaScript code from accessing the container page unless the iframe comes from the same domain. Since the embedded SIDGrid shindig is running on the 8080 port as a Tomcat web application, the gadget has to rely on the OpenSocial's gadget-to-container RPC mechanism to implement the cross-domain communication with the SIDGrid container page.

4.2 SIDGrid Collaboration Based on Social Networking

OpenSocial provides the social data API for gadget developers to build social networking applications. This social data API is defined on the model < person, group, activity, appdata >, which can describe the profile information for a person or the relationship between people or activities. Web gadgets can retrieve the social data through the API to implement social applications. Modern research projects often involve multiple scientists from different academic institutions. Social applications can greatly facilitate such teamwork in terms of data sharing, collaborative data analysis, and knowledge discoveries.

Based on the social data model, we introduce social networking into the SIDGrid to enable the development of socially aware gadgets for better collaboration and data sharing among SIDGrid users. In the existing SIDGrid implementation, a group security model allows users to create a UNIX-style group and select their collaborators from the SIDGrid user list to add to the group for data sharing. This group feature can be easily extended into the OpenSocial space. For example, suppose SIDGrid user A wants to share his experiment with his team members including user B and user C. User A establishes a SIDGrid group for his experiment and adds B and C to the group so that B and C can access the experiment. In this scenario, the group relationship between A, B, and C can be naturally described as an OpenSocial group. Any computational workflows performed by A, B, or C to process the experiment should become visible to all of them. Similarly, any data update on the experiment made by A, B, or C should be notified as an OpenSocial activity in the group.

Most features in the OpenSocial data API have been developed in the OpenSocial reference implementation Shindig. To gadget developers, it is a group of RESTful and JSON-RPC services that expose the social data graph in the database of the server. Ideally we just need to add the database connector code into the Shindig package to query the user and group information in the SIDGrid database. However, Shindig does not implement the Group according to the OpenSocial specification in the general sense but only provides the special group named "Friend" for gadgets. Apparently the "Friend" group is too general to describe specific data sharing groups in the SIDGrid since any two SIDGrid users that have a common group can be regarded as friends. Therefore we extend the implementation of the social data part in Shindig and add more features on the SIDGrid groups and activities to the JavaScript API that is available to gadgets.

The RESTful API for the SIDGrid social data features is shown in Table 1.

Table 1. SIDGrid Social Data API

Type	RESTful URL	Explanation
Group	/people /{userid} /@all-groups	List all the group for this SIDGrid user
	/people /{userid} /{groupid}	List all the people connected to the user {userid} in group {groupid}
Activities	/activities /{guid} /@self	Collection of activities generated by given user
	/activities/{guid} /@self/{appid}	Collection of activities generated by an app for a given user
	/activities /{guid} /{groupid}	Collection of activities generated by an app for a given user in a given group
	/activities/{guid} /{groupid} /{appid}	Collection of activities generated by an app for people in group {groupid} belonging to given user {uid}
Work flows	/workflow /{userid}	List all the workflows created by the user {userid}
	/workflow /{userid} /{appid}	List all the workflows of the application {appid} created by the user {userid}
	/workflow /{userid} /{groupid}	List all the workflows created by the users for the experiments in the group {groupid} in which the user {userid} joins
	/workflow /{userid} /{groupid} /{appid}	List all the workflows of the application {appid} created by the users for the experiments in the group {groupid} in which the user {userid} joins
	/workflow /{userid}/{expid}	List all the workflows created by the users for the experiment {expid}
Experiments	/experiment /{userid}	List all the experiments created by the user (userid)
	/experiment /{userid} /{groupid}	List all the experiments of the group {groupid}

Collaborative application gadgets can be developed based on the SIDGrid social data API. For instance, if members of a research team share their experimental data in the SIDGrid, they would like to be aware of any data update or analytical workflows performed on the sharing dataset. Doing so requires a collaborative workflow browsing gadget that can tell the users not only about their own workflows but also about the workflows run by their teammates. Figure 9 displays such a gadget running on a SIDGrid web page (myWorkflows) under the user "Susanne," showing workflows created by two users in the group of "wwjtest."

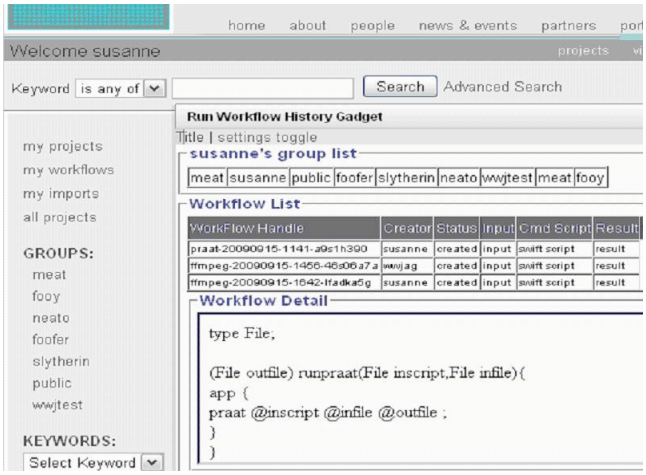

Figure 9. Collaborative workflow history gadget.

5. Conclusion

In this paper, we describe a new Web 2.0-based application framework of the SIDGrid, which can integrate commonly used social and behavioral science tools and provide researchers a collaborative web interface to run these data-intensive applications efficiently on the TeraGrid resources. We have developed a new SIDGrid science gateway based on this framework and have deployed it on the SIDGrid server. A few application gadgets have been built through the application framework and are available to SIDGrid users. We believe that with this simplified mechanism for application creation and deployment, the new SIDGrid science gateway can greatly extend the class of automated analysis experiments that can be conducted by social and behavioral scientists. We anticipate that this new gateway will enable these scientists to efficiently conduct near-real-time analysis of experimental data.

Acknowledgments

This work was supported by the Office of Advanced Scientific Computing Research, Office of Science, U.S. Department of Energy, under Contract DE-AC02-06CH11357, and by the National Science Foundation under grant OCI-0504086.

References

[1] Bertenthal, B., Grossman, R, Hanley, D., Hereld, M, Kenny, S., Levow, G., Papka, M., Porges, S., Rajavenkateshwaran, K., Stevens, R., Uram, T., and Wu, W. 2007. Social Informatics Data Grid, E-Social Science 2007 Conference, October 7-9, 2007, Ann Arbor, Michigan.

[2] Kandaswamy, G., Fang, L., Huang, Y., Shirasuna, S., Marru, S., and Gannon, D. 2006. Building Web Services for Scientific Grid Applications. IBM Journal of Research and Development 50(2-3).

[3] Krishnan, L., Stearn and B., Bhatia, Baldridge, K.K., Li, W.W., and Arzberger, P. 2006. In Proc. IEEE International Conference on Web Services (ICWS 2006), pp. 823-832.

[4] OpenSocial Specification, http://www.opensocial.org/

[5] Mobyle, http://bioweb2.pasteur.fr/projects/mobyle/

[6] Zhao, Y., Hategan, M., Clifford, B., Foster, I., vonLaszewski, G., Raicu, I., Stef-Praun, T., and Wilde, M. 2007. Swift: Fast, Reliable, Loosely Coupled Parallel Computation. In Proc. IEEE International Workshop on Scientific Workflows, pp. 199-206.

[7] Shindig: http://incubator.apache.org/shindig/

[8] Boersma, P. 2001. Praat, a system for doing phonetics by computer. Glot International 5(9-10) 341-345.

[9] OAuth, http://oauth.net/core/1.0

[10] Welch1, V., Barlow, J., Basney, J., and Marcusiu, D. 2006. AAA model to support science gateways with community accounts. Concurrency and Computation: Practice and Experience 19(6) 893-904